AD-A038 210 TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES F/G 12/1
ENHANCEMENTS OF SPANNING TREE LABELING PROCEDURES FOR NETWORK O--ETC(U)
DEC 76 R BARR, F GLOVER, D KLINGMAN N00014-75-C-0616
UNCLASSIFIED CCS-262 NL

1 OF 1
AD A038210

END
DATE FILMED
4-77

13

1473

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

(12)

Research Report CCS 262

ENHANCEMENTS OF SPANNING TREE
LABELING PROCEDURES FOR NETWORK OPTIMIZATION

by

Richard Barr*
Fred Glover**
Darwin Klingman***

December 1976

*Assistant Professor of Management Science and Computers, Southern Methodist University, School of Business Administration, Dallas, TX 75275

**Professor of Management Science, University of Colorado, Boulder, CO 80302

***Professor of Operations Research, Statistics, and Computer Science and Director of Computer Science Research, BEB-608, University of Texas at Austin, Austin, TX 78712

This research was partly supported by Project NR047-021, ONR Contracts N00014-75-C-0616 and N00014-75-C-0569 with the Center for Cybernetic Studies, The University of Texas. Reproduction in whole or in part is permitted for any purpose of the United States Government

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 203E
The University of Texas
Austin, Texas 78712
(512) 471-1821

ABSTRACT

New labeling techniques are provided for accelerating the basis exchange step of specialized linear programming methods for network problems. Computational results are presented which show that these techniques substantially reduce the amount of computation involved in updating operations.

1.0 INTRODUCTION

In solving minimum cost flow network problems by specialized linear programming methods, an important question is: How can one update the spanning tree basis with the least amount of effort? A partial answer to this question is provided by special list structure techniques such as the API method [6] and the more recent ATI method [9], which have contributed dramatically to improving the efficiency of network algorithms (see, e.g., [3, 5, 6, 7, 11]). This paper addresses the issue of which supplemental techniques can be used to enable these list structures (and particularly the ATI method) to be implemented with greater efficiency.

As shown in [6], the major updating calculations of a basis exchange step can be restricted to just one of the two subtrees created by dropping the outgoing arc. Consequently, a natural goal is to identify the smaller of these two subtrees by means of a function $t(x)$ that names the number of nodes in the subtree "headed by node x." A clever and rather intricate procedure for doing this was proposed by Srinivasan and Thompson in [13]. Unfortunately, this procedure requires sorting the nodes of the subtree by their distances from the root, and then further entails a full subtree update of both the distance values and the $t(x)$ values at each basis exchange step. Because of the substantial amount of work required to update the $t(x)$ list, the advantages of using this list have been largely offset by the computational costs involved in its maintenance, and the potential of the original Srinivasan-Thompson proposal has not been fully realized.

The purpose of this paper is to propose a new type of relabeling scheme that succeeds in updating t(x) without sorting. In fact, this scheme requires even less work than to update the distance values of [13]. The relabeling is based on "absorbing" t(x) into the updating calculations of the ATI method. Moreover, these calculations are carried out simultaneously with the procedures [9] for updating other changes introduced by the basis exchange step.

To achieve the integration of the ATI calculations and the update of t(x), an index function f(x) is introduced that names the last node in the subtree rooted at x. Additionally, it is shown that f(x) makes it possible to streamline the ATI calculations. Finally, as a bonus, it is shown that t(x) can accommodate all of the relevant functions filled by the distance values, and hence can replace these values. Computational results presented in the last section indicate that the net gains of all these advantages produce a substantially improved procedure for implementing the basis exchange operations.

2.0 NOTATION AND DEFINITIONS

It will be assumed that a basis tree with n nodes and n - 1 arcs is known and has been rooted. The root node will be regarded as the "highest" node in the rooted tree with all other nodes hanging below it. If nodes i and j denote endpoints of a common arc in the rooted tree such that node i is closest to the root, then i is called the *predecessor* of node j and node j is called an *immediate successor* of node i.

The following notational conventions will be used to identify the components of the basis exchange step:

(p,q) = the arc leaving the basis, where p is currently the predecessor of q.

(u,v) = the arc entering the basis, where u is the node whose unique path to the root node contains arc (p,q).

T = the basis tree.

T(x) = the subtree of T that is rooted at node x (hence the subtree that includes x and all its successors under the predecessor ordering).

p(x) = the predecessor of node x where p(x) = 0 if node x is the root node.

s(x) = the "thread successor" of x.

Intuitively, function s may be thought of as a thread which passes through each node exactly once in a top to bottom, left to right order starting from the root node.

More precisely the function s satisfies the following inductive characteristics:

(a) Letting 1 denote the root node, the set $\{1, s(1), s^2(1), \ldots s^{n-1}(1)\}$ is precisely the set of nodes of the rooted tree where $s^2(1) = s(s(1))$, $s^3 = s(s^2(1))$, etc. The nodes $1, s(1), \ldots s^{k-1}(1)$, will be called the *antecedents* of node $s^k(1)$.

(b) For each node i other than node $s^{n-1}(1)$, s(i) is one of the immediate successors of node i, if i has a successor. Otherwise, s(i) is an immediate successor of the closest predecessor of node i, say node x, such that node x has an immediate successor which is not an antecedent of node i.

(c) $s^n(1) = 1$; that is, the last node of the tree threads back to the root node.

By virtue of the foregoing characterization, the set of nodes of $T(x)$ is $\{x, s^1(x), \ldots, s^k(x)\}$ where k is the largest number such that $p(s^k(x))$ is one of the nodes $x, s^1(x), \ldots, s^{k-1}(x)$. By convention $s^1(x) = s(x)$ and $s^0(x) = x$.

$t(x)$ = the number of nodes in $T(x)$.

$f(x)$ = the "last node," $s^r(x)$, of the nodes in $T(x)$, where $r = t(x) - 1$.

Figure 1 illustrates the above functions as follows. The NODE array specifies the node names. The entries in the arrays p, s, f, and t parallel to a node name specify the values of the functions p, s, f, and t for that node name. Note that the direction of the arcs in Figure 1 corresponds to the direction induced by the predecessor ordering and may not correspond to the direction of the arc in the underlying problem.

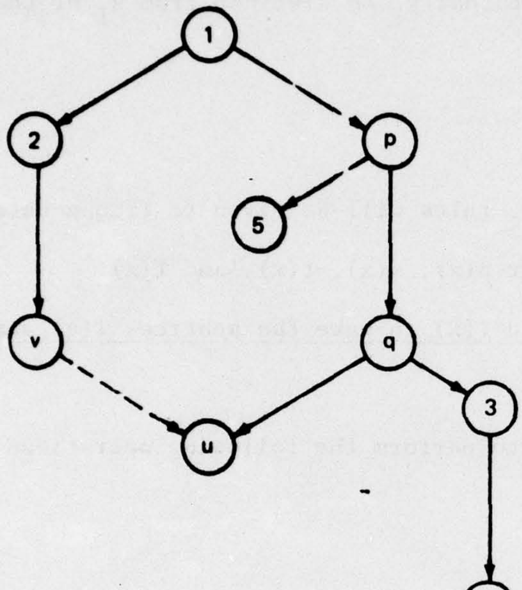
The basis exchange step may be visualized as consisting of two components:

(1) Dropping arc (p,q) to create two independent subtrees: T(q) and T-T(q) (where the latter is the subtree of T that excludes T(q) and all its nodes, and hence which excludes the "connecting arc" (p,q));

(2) Adding arc (u,v) to create a single new basis tree.

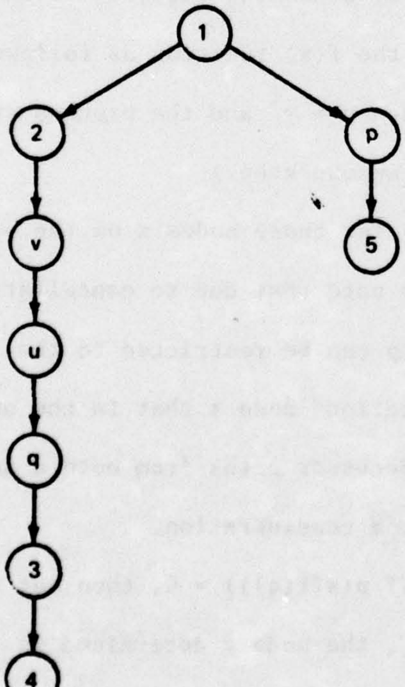
The subtrees T(q) and T-T(q) can be viewed as any two node-disjoint trees which are to be joined by an arc to create a new basis tree. Updating operations will be developed first to make T(q) and T-T(q) into label "independent" trees in preparation for selecting which is to be the new "upper tree" (called T_1) rooted at x_1 and which is to be the new "lower tree" (called T_2) rooted at x_2. It may then be assumed that the root of T_1 becomes the root of the new basis tree. Additionally, (y_1, y_2) will be used to denote the arc that joins T_1 and T_2 where y_1 is a node of T_1 and y_2 is a node of T_2. Next, operations will be developed to re-root T_2 at

Figure 1



NODE	p	s	f	t
1	0	2	4	9
2	1	v	v	2
3	q	4	4	2
4	3	1	4	1
5	p	q	5	1
p	1	5	4	6
q	p	u	4	4
v	2	p	v	1
u	q	3	u	1

Figure 2



NODE	p	s	f	t
1	0	2	5	9
2	1	v	4	6
3	q	4	4	2
4	3	p	4	1
5	p	1	5	1
p	1	5	5	2
q	u	3	4	3
v	2	u	4	5
u	v	q	4	4

y_2 in preparation for attaching T_2 to T_1 via arc (y_1, y_2) to create the new basis tree. (There is no requirement that y_1 be distinct from x_1 or that y_2 be distinct from x_2.)

3.0 UPDATING OPERATIONS

Using the preceding definitions, rules will be given to find updated values p*(x), s*(x), t*(x), f*(x) for p(x), s(x), t(x), and f(x).

I. Update p(x), s(x), t(x), and f(x) to make the subtrees T(q) and T-T(q) independent.

The reader may find it helpful to perform the following operations using Figure 1.

I.1. Update for T-T(q):

For p*(x): No updating of any p(x) is required.

For s*(x): Identify the node y in T-T(q) such that s(y) = q. Then set s*(y) = s(f(q)). No other s(x) values are changed. (Note: While this step is obviously facilitated by directly accessing f(q), the identification of y is further speeded by utilizing the f(x) function as follows: First, let $y^1 = p$. Second, if $s(y^1) = q$, then $y = y^1$ and the process stops. Otherwise, let $y^1 = f(s(y^1))$ and repeat the second step.)

For t*(x): Set t*(x) = t(x) - t(q) for those nodes x on the path from p to the root of T. It is important to note that due to cancellation effects of subsequent calculations, this step can be restricted to the partial backward path from p to the "intersection" node z that is the unique node of the basis loop that lies on the predecessor paths from both u and v, to the root of T, excluding node z itself from consideration.

For f*(x): Let x* = p(s(f(q))). If p(s(f(q))) = 0, then set x* equal to the root node. Set f*(x) = y (i.e., the node y determined in

updating s above) for those nodes x on the path from p to x*, excluding x* itself if $p(s(f(q))) \neq 0$. (If x* = p, then no updating is done unless x* is the root node.)

I.2. Update for T(q):

For p*(x): Set p(q) = 0

For s*(x): Set s*(f(q)) = q.

No other updating of any p(x), s(x), t(x), or f(x) values is required for T(q).

II. Decide which of T(q) and T-T(q) is to be T_1 and which is to be T_2.

The rule to use for picking which subtree to re-root heavily depends on when the dual variables are updated. If the updating of the dual variables is not integrated with the other updating operations, it will be clear from the operations performed in Step III that the subtree to re-root should be selected according to the number of nodes in the predecessor paths from v to the root of T-T(q) and from u to q. In particular, the subtree associated with the path containing the minimum number of nodes should be re-rooted. Further, if the updating operations are not integrated, the dual variables should be updated in the subtree containing the smallest number of nodes.

Thus the following procedures will be computationally investigated in section 5:

Procedure 1: Let T_2 be the subtree whose predecessor path from the proposed new root to the old root is smallest and separately update the dual variables in the subtree containing the fewer number of nodes.

Procedure 2: Same as procedure 1 except that if T_2 is to be both re-rooted and its dual variables updated, then integrate the updating of the

dual variables with the other updating operations.

One computational disadvantage of the above procedures is that the number of nodes in the path from u to the root of T-T(q) is not known and to calculate it involves traversing the path from the intersection node z to the root of T-T(q) for no purpose other than to calculate this number. Another computational disadvantage simply involves the fact that there are certain computational advantages to always performing all updating operations on one subtree. To partially overcome these difficulties, a compromise procedure is proposed:

Procedure 3: If t(q) exceeds n/2 let T(q) be T_1 and let T-T(q) be T_2 (hence x_1 = q and x_2 = the root of T). Otherwise, let T-T(q) be T_1 and let T(q) be T_2. Perform all updating operations on T_2 integrating the dual variable updating with the other updating operations.

Procedure 4: Let T_2 be the subtree containing the smallest number of nodes.

III. Make y_2 the new root of T_2 and update p(x), s(x), t(x), and f(x).

It is possible to combine the updating of the functions in several computationally efficient ways, for it is possible to simultaneously update all of the functions, while making exactly one traversal of the nodes in the unique predecessor path from y_2 to x_2 and their descendants. However, for readability the updating of each function will be presented separately assuming that no other functions have been updated since step I. Thus the order of the updating procedures given below is arbitrary. In the following, p(x), f(x), t(x), and s(x) refer to the value of the functions after the execution of step I. (Again, the reader may find it helpful to carry out these operations using an updated Figure 1 and procedure 3 from section II.)

For $p^*(x)$: Reverse the predecessor orientation of the path from y_2 to x_2. That is, if $x_2 = y_2$, do nothing. Otherwise, recursively set $p^*(p(y)) = y$ and then $y = p(y)$ until and including $p(y) = x_2$ initially setting $y = y_2$. Finally set $p^*(y_2) = 0$. (It is important to note that $p(x)$ and $p^*(x)$ must be kept distinct from each other; i.e., it is not legitimate to replace $p(x)$ by $p^*(x)$ in the computation.) No other updating of $p(x)$ is required.

For $t^*(x)$: Set $t^*(y_2) = t(x_2)$. (But if $T(q) = T_1$, and the restricted update of $t(x)$ was carried out in step I.1, set $t^*(y_2) = t(x_2) - t(q)$.) Then for each x on the predecessor path from y_2 to x_2 (excluding $x = x_2$), set $t^*(p(x)) = t^*(y_2) - t(x)$. (Again note that $t(x)$ and $t^*(x)$ must be kept distinct and it is assumed that the function $p(x)$ has not been updated.) No other updating of $t(x)$ is required.

For $s^*(x)$: If $x_2 = y_2$ no updating of $s(x)$ is required. Otherwise execute the following:

i) Set $x = y_2$, $w = f(x)$, and $z = s(w)$.

ii) Find the unique node y such that $s(y) = x$. (Note that y may be found as in step I.1 by letting $y^1 = p(x)$.)

iii) If $p(z) = p(x)$, set $s^*(y) = z$, $s^*(w) = p(x)$, $w = f(p(x))$, $z = s(w)$, and go to step iv. Otherwise set $s^*(w) = p(x)$, $w = y$ and proceed.

iv) Set $x = p(x)$.

v) If $x \neq x_2$ go to step ii. Otherwise set $s^*(w) = y_2$ and stop. In the updating of f, this last value of w (i.e., the node whose new thread is the new root y_2) plays a primary role. (Note that w is either equal to $f(x_2)$ or the last y.)

<u>For $f^*(x)$</u>: If $x_2 = y_2$ no updating is required. Otherwise, let $\bar{x}_2$ be the unique node on the predecessor path from y_2 to x_2 such that $p(\bar{x}_2) = x_2$. If $f(x_2) \neq f(\bar{x}_2)$ then $f^*(x_2) = f(x_2)$ (i.e., $f(x_2)$ does not change). Otherwise, $f^*(x_2) = y$ where y is the unique node such that $s(y) = \bar{x}_2$. (This y may be found as in I.1; but as noted above, it will be identified automatically at the conclusion of updating s.) In either case, set $f^*(x) = f^*(x_2)$ for all x on the predecessor path from y_2 to x_2. No other changes are made.

It appears that in order to optimize computational operations, step III should be implemented as follows. One should traverse the path from y_2 to x_2 simultaneously updating the functions $p(x)$, $s(x)$, and $t(x)$, the basic flow values associated with arcs on this path, and co-ordinating the updating of the node potential values with the updating of $s(x)$. Next the new predecessor path from x_2 to y_2 should be traversed to update $f(x)$.

IV. <u>Attach T_2 to T_1 by adding arc (y_1,y_2) to create the new basis tree (where T_2 is now rooted at y_2 as a result of step III).</u>

As before, in the following $p(x)$, $s(x)$, $f(x)$, and $t(x)$ refer to the value of the functions p, s, f, and t after the execution of step III above and the rules for updating each function assume that no other functions have been updated since step III.

<u>For $p^*(x)$</u>: Set $p^*(y_2) = y_1$.

<u>For $s^*(x)$</u>: Set $s^*(f(y_2)) = s(y_1)$ and $s^*(y_1) = y_2$.

<u>For $t^*(x)$</u>: Set $t^*(x) = t(x) + t(y_2)$ for all x on the path from y_1 to x_1. (But if $T(q) = T_2$, and the restricted update of $t(x)$ was applied

in step I.1, then the current step should be restricted to those x on the path from y_1 to z, excluding z itself, for the "intersection" node z as identified in I.1.)

For $f^*(x)$: Set $\bar{x} = p(s(y_1))$. If $\bar{x} = 0$ set $\bar{x} = x_1$. Then for those nodes x on the backward path from y_1 to $\bar{x}$, excluding $\bar{x}$ itself if $p(s(y_1)) \neq 0$, set $f^*(x) = f(y_2)$.

(Figure 2 indicates the results of applying steps I, II, III, and IV above on Figure 1.)

The proposed procedure for updating t(x) clearly requires less effort than updating the distance function of [13], which involves an addition for every node of the subtree T(q), and in the case of $T(q) = T_1$, requires an addition for every node of T. The fact that t(x) can replace the distance function is a direct consequence of the observation that t(x) can be used in essentially the same manner as the distance function to facilitate operations of identifying the loop created by adding arc (u,v) to the basis tree. Specifically this loop can be located as follows:

i) Set $x_u = u$ and $x_v = v$.

ii) If $t(x_u) < t(x_v)$ go to step v.

iii) If $t(x_u) > t(x_v)$ go to step vi.

iv) If $x_u \neq x_v$, go to step v. Otherwise stop. The loop created by adding (u,v) has been traversed and $x_u = x_v = z$ is the unique node of this loop which lies on the predecessor paths from both u and v to the root node of the tree.

v) Search the predecessor path of u starting at x_u for a node x such that $t(x) \geq t(x_v)$ and set $x_u = x$. If $t(x_u) = t(x_v)$ go to step iv. Otherwise go to step vi.

vi) Search the predecessor path of v starting at x_v for a node x such that $t(x) \geq t(x_u)$ and set $x_v = x$. If $t(x_u) = t(x_v)$ go to iv. Otherwise go to step v.

4.0 INITIALIZATION

It is left to characterize the procedure for establishing the initial values of t(x) and f(x). This occurs simultaneously with the initial determination of the s(x) values as follows.

Let x_o denote the root of the tree. Consider the step in which $s^{k+1}(x_o) = s(s^k(x_o))$ is identified $(k \geq 0)$. If $s^k(x_o)$ is the predecessor of $s^{k+1}(x_o)$ (via the predecessor indexing), do nothing. Otherwise, for all nodes $s^i(x_o)$ on the predecessor path from $s^k(x_o)$ to the predecessor of $s^{k+1}(x_o)$, excluding the predecessor of $s^{k+1}(x_o)$ itself, set $t(s^i(x_o)) = k + 1 - i$, and set $f(s^i(x_o)) = s^k(x_o)$.

When the last node $s^{n-1}(x_o)$ of the network is determined, set $t(s^i(x_o)) = n - i$ and set $f(s^i(x_o)) = s^{n-1}(x_o)$ for all $s^i(x_o)$ on the predecessor path from $s^{n-1}(x_o)$ to x_o.

To easily keep track of the index i for each node $s^i(x_o)$ that is to be considered on a given step, it is convenient to keep a list that consists precisely of the indexes i of the nodes $s^i(x_o)$ to x_o. Specifically, to begin with the list contains the single index 0 (for $s^0(x_o)$). When $s^{k+1}(x_o)$ is created, the number k + 1 is added to the end of the list. When a predecessor path from $s^k(x_o)$ is traced, consisting of r nodes (say) $s^i(x_o)$ whose values $t(s^i(x_o))$ and $f(s^i(x_o))$ are to be set, the indexes of these r nodes will be exactly the corresponding last r numbers on the list.

By removing these numbers from the list just before adding the number k + 1, the desired structure of the list is maintained.

5.0 COMPUTATIONAL ANALYSIS

5.1 Highlights of the Development of the ARC-II Computer Code

To evaluate the foregoing procedures, henceforth referred to as the Extended Threaded Index (XTI) Method, we developed a new in-core computer code entitled ARC-II for solving capacitated transshipment problems. ARC-II is written in a "manilla" FORTRAN with several subroutines to perform the various updating operations, for the following reasons: (1) this modular approach simplifies testing of different updating procedures, (2) minimal recoding is required to fit different machine and computer conventions, and (3) unbiased comparisons can be made with codes which have not been "customized" to a particular machine or compiler. One disadvantage of this approach, of course, is that the reported times are conservative, since programs which have been "tuned" to a particular operating environment execute substantially faster. However, our purpose in developing ARC-II was to obtain unbiased comparisons between the XTI approach and other procedures available in the literature. To this end, the same starting and pivoting procedures as described in [8, 9] for transshipment problems are used.

After initially developing ARC-II, preliminary testing was conducted on the recoding rules described in part II of section 3. Our initial testing indicated that procedure 3 was never a good rule and that procedure 4 marginally dominated procedures 1 and 2. Thus, the times on the ARC-II code reported subsequently in this section reflect the use of procedure 4.

Another supplementary feature tested was an "inverse thread function." Using this function, in conjunction with the other functions previously discussed, one can eliminate all searching involved in the basis exchange operation; i.e., updating the thread function is simply a matter of resetting known pointers. The disadvantages of using the inverse thread include increased memory requirements and the need to maintain an additional set of function values. Our tests using the inverse thread function indicate that solution times are reduced by approximately 5%. In our opinion, this reduction is not sufficient to warrant the utilization of additional memory space, and therefore, ARC-II does not use such a function.

5.2 Computational Comparisons

To determine the efficiency of the XTI procedures, ARC-II was compared with three out-of-kilter codes referred to hereinafter as SHARE [4], SUPERK [1], and BSRL (developed by T. Bray and C. Witzgall while at Boeing Scientific Laboratories). Additionally, one dual simplex based code [5], called DNET, one negative cycle code [2], called BENN, and two primal simplex based codes, called PNET, PNET-I, were tested for comparative purposes. PNET [8] uses the API list structure [6] and PNET-I [9] uses the ATI list structure [9].

All of the above mentioned codes are in-core codes; i.e., the program and all of the problem data simultaneously reside in fast-access memory. All are coded in FORTRAN and none of them (including ARC-II) have been tuned (optimized) for a particular compiler. All of the problems were solved on the CDC 6600 at the University of Texas Computation Center using

the RUN compiler. The computer jobs were executed during periods when the machine load was approximately the same, and all solution times are exclusive of input and output; i.e., the total time spent solving the problem was recorded by calling a Real Time Clock upon starting to solve the problem and again when the solution was obtained.

Since the test problems of [11] are currently used worldwide for comparison purposes, they were also used in our comparison. This comparison included several different types of problems (e.g., assignment, transportation, and minimum cost flow network problems), both capacitated and uncapacitated, and with varying node and arc requirements. The problem specifications of these 35 problems as required on the input cards to the network generator are given in [11]. Problems 1-5 are 100 x 100 transportation problems; problems 6-10 are 150 x 150 transportation problems. Problems 11-15 are 200 x 200 assignment problems. Problems 16-27 are 400 node capacitated transshipment problems; problems 28-35 are uncapacitated 1000 and 1500 node transshipment problems. Table I reports the solution times for each of these problems for each of the codes.

The results in Table I clearly indicate the superiority of ARC-II over all other codes tested. Furthermore, the data indicate, rather startlingly, that ARC-II is approximately twice as fast as one of the (previously) fastest codes in the literature, PNET-I.

It is also particularly noteworthy that the solution times for the ARC-II code are based on using the simple pivot strategies of [8], rather than the more sophisticated candidate list strategies whose superiority has been documented by Mulvey [12]. We have used the simpler pivot strategies to make it possible to more clearly differentiate the contribution

TABLE I

Solution Times in Seconds on a CDC 6600

Problems	ARC-II	PNET	PNET-I	DNET	BENN	SUPERK	SHARE	BSRL
1	.78	1.30	1.07	12.85	20.25	5.68	17.76	30.25
2	.89	1.49	1.25	13.56	24.36	6.47	21.34	21.59
3	1.01	1.94	1.64	21.44	34.56	6.87	26.16	31.47
4	.95	1.64	1.27	17.96	31.45	6.57	25.13	36.47
5	1.25	1.88	1.63	23.34	52.10	6.77	30.97	47.73
6	2.11	3.55	2.86	46.10	61.00	11.05	46.40	46.64
7	2.23	4.06	3.37	74.88	DNR	12.86	65.92	113.12
8	2.99	4.72	4.10	97.92	DNR	13.69	81.00	175.10
9	2.99	4.80	4.15	101.65	DNR	13.40	81.21	186.99
10	4.02	5.88	5.27	95.96	DNR	14.13	84.24	184.75
11	1.92	3.52	2.31	19.87	17.44	6.44	19.93	30.39
12	2.36	4.87	3.71	26.58	20.31	6.47	21.17	22.08
13	3.13	5.52	3.47	27.98	24.92	7.25	25.81	20.02
14	2.96	6.02	3.44	30.15	27.40	6.95	24.95	23.11
15	3.12	6.50	4.79	31.57	DNR	7.56	27.05	21.08
16	1.38	2.40	2.15	14.77	11.77	5.27	21.51	15.05
17	1.87	3.11	2.60	DNR	20.10	8.36	32.40	64.64
18	1.26	1.92	1.70	DNR	11.31	5.13	20.06	18.31
19	1.72	2.60	2.40	DNR	20.62	8.49	31.75	61.07
20	1.28	2.67	2.47	DNR	10.38	4.69	18.11	25.72
21	1.83	2.76	2.46	DNR	20.35	7.96	32.60	61.39
22	1.26	2.22	2.01	DNR	9.97	4.60	17.91	24.84
23	1.67	3.00	2.74	DNR	19.81	7.91	32.66	67.96
24	1.52	3.12	2.91	DNR	11.11	5.59	25.27	21.57
25	1.83	4.17	3.96	DNR	18.27	8.37	33.19	48.40
26	1.08	4.45	4.05	DNR	11.38	5.51	25.05	19.34
27	1.62	4.42	4.21	DNR	16.37	7.50	30.45	41.98
28	4.40	6.35	5.37	DNR	DNR	13.91	53.87	83.98
29	4.87	7.39	6.25	DNR	DNR	14.51	52.55	117.83
30	4.88	9.08	7.90	DNR	DNR	16.00	61.33	152.21
31	5.68	9.59	7.58	DNR	DNR	17.05	61.33	135.73
32	7.42	15.70	11.73	DNR	DNR	22.88	78.63	553.93
33	7.82	20.20	15.95	DNR	DNR	25.89	101.92	210.14
34	8.21	17.10	13.76	DNR	DNR	25.42	92.25	248.16
35	8.81	19.39	15.87	DNR	DNR	29.96	DNR	DNR

***DNR--Did not run.**

of the new labeling procedures. (There has indeed been some confusion introduced into the literature by a number of recent studies whose comparisons have not been based on fundamental methodological differences in labeling procedures, but simply on differences in pivot strategies, not clearly identified as such.) Thus, the timesin Table I, while extremely fast, should not be construed as the best attainable with the ARC-II code. Preliminary tests with candidate list strategies, not yet refined to achieve the most effective trade-offs with the new labeling procedures, have in fact resulted in times that are roughly half of those in Table I.

5.3 Memory Requirements of the Codes

Table II indicates the number of node and arc length arrays required in each of the codes tested for solving capacitated problems. It should be noted that the program memory requirements of all of the codes tested were quite close (within 1000 words) excluding the array requirements. Thus the important factor in comparing codes is the number of node and arc length arrays. Also, it should be noted that the primal and dual simplex codes require one less arc length array if the problem is uncapacitated. This is not true of the out-of-kilter code.

Since any meaningful network problem has to have more arcs than nodes, it is clear by Table II that the primal and dual simplex codes have a distinct advantage (in terms of memory requirements) over all of the other codes. Further, this advantage greatly increases as the number of arcs increase and if the problem is uncapacitated. For example, for a

TABLE II

CODE SPECIFICATIONS

	Developer	Name	Type	Number of Arrays
1.	Barr, Glover, Klingman	ARC-II	Primal Simplex Network	7N + 3A
2.	Barr, Glover, Klingman	SUPERK	Out-of-kilter	4N + 9A
3.	Bennington	BENN	Non-simplex	6N + 11A
4.	Bray and Witzgall	BSRL	Out-of-kilter	6N + 8A
5.	Clasen	SHARE	Out-of-kilter	6N + 7A
6.	Glover, Karney, Klingman	PNET	Primal Simplex Network	7N + 3A
7.	Glover, Karney, Klingman	DNET	Dual Simplex Network	9N + 3A
8.	Glover, Karney, Klingman	PNET-I	Primal Simplex Network	6N + 3A
9.	General Motors	GM	Out-of-kilter	3N + 6A

N - Node Length
A - Arc Length

problem which has 10 times as many arcs as nodes, ARC-II, PNET, PNET-I, or DNET require only about one-half the memory space of the best of the other codes.

ACKNOWLEDGEMENTS

A number of particularly apt comments by a referee have improved the readability of this paper and are sincerely acknowledged. Also, the editorial assistance of Dr. John Hultz, Systems Analyst for Analysis, Research, and Computation, Inc., and Mr. David Karney, Systems Analyst, Center for Cybernetic Studies, are appreciated.

The authors also wish to acknowledge the cooperation of the staff of The University of Texas Business School Computation Center, and Southern Methodist University Computation Center.

REFERENCES

1. Barr, R. S., F. Glover, and D. Klingman, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," *Mathematical Programming*, 7, 1, pp. 60-87, 1974.

2. Bennington, G. E., "An Efficient Minimal Cost Flow Algorithm," O. R. Report 75, North Carolina State University, Raleigh, North Carolina, June 1972.

3. Bradley, G., G. Brown, and G. Graves, "A Comparison of Storage Structure for Primal Network Codes," presented at ORSA/TIMS conference, Chicago, April 1975.

4. Clasen, R. J., "The Numerical Solution of Network Problems Using the Out-of-Kilter Algorithm," RAND Corporation Memorandum RM-5456-PR, Santa Monica, California, March 1968.

5. Glover, F., D. Karney, and D. Klingman, "Double-Pricing Dual and Feasible Start Algorithms for the Capacitated Transportation (Distribution) Problem," CCS Research Report 105, Center for Cybernetic Studies, The University of Texas, Austin, Texas 78712.

6. Glover, F., D. Karney, and D. Klingman, "The Augmented Precedessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," *Transportation Science*, 6, 171-180 (1972).

7. Glover, F., D. Karney, D. Klingman, and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Management Science*, 20, 5, 793-813 (1974).

8. Glover, F., D. Karney, and D. Klingman, "Implementation and Computational Study on Start Procedures and Basis Change Criteria for a Primal Network Code," *Networks*, 20, 191-212 (1974).

9. Glover, F., D. Klingman, and J. Stutz, "Augmented Threaded Index Method," *INFOR*, 12, 3, 293-298 (1974).

10. Johnson, E., "Networks and Basic Solutions," *Operations Research*, 14, 4, 619-623 (1966).

11. Klingman, D., A. Napier, and J. Stutz, "NETGEN - A Program for Generating Large Scale (Un)Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," *Management Science*, 20, 5, 813-819 (1974).

12. Mulvey, J., "Column Weighting Factors and Other Enhancements to the Augmented Threaded Index Method for Network Optimization," Joint ORSA/TIMS Conference, San Juan, Puerto Rico, 1974.

13. Srinivasan, V. and G. L. Thompson, "Accelerated Algorithms for Labeling and Relabeling of Trees with Application for Distribution Problems," *Journal of the Association for Computing Machinery*, 19, 4, 712-726 (1972).

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Center for Cybernetic Studies The University of Texas	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

6

3. REPORT TITLE

Enhancements of Spanning Tree Labeling Procedures for Network Optimization.

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

9 Research rept.

10

5. AUTHOR(S) (First name, middle initial, last name)

15 N00014-75-C-0616, N00014-75-C-0569

tp

Richard Barr

Fred Glover

Darwin Klingman

11 6. REPORT DATE Dec[redacted] 1976	7a. TOTAL NO. OF PAGES 23	7b. NO. OF REFS 13

8a. CONTRACT OR GRANT NO. N00014-75-C-0569; 0616 b. PROJECT NO. NR047-021 12 26p. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Center for Cybernetic Studies Research Report CCS-262 14 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Office of Naval Research (Code 434) Washington, D.C.

13. ABSTRACT

New labeling techniques are provided for accelerating the basis exchange step of specialized linear programming methods for network problems. Computational results are presented which show that these techniques substantially reduce the amount of computation involved in updating operations.

406 197

DD FORM 1 NOV 65 1473 (PAGE 1)

Unclassified

LB

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Networks						
Spanning Trees						
Linear Programming						
Link Lists						
Maximum Flow						
Transportation						
Network Flows						